PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

PROGRAMACIÓN 3
2da práctica (tipo b)
(Primer Semestre 2025)

Indicaciones Generales:
Duración: **110 minutos.**

- Se les recuerda que, de acuerdo al reglamento disciplinario de nuestra institución, constituye una falta grave copiar del trabajo realizado por otro estudiante o cometer plagio para el desarrollo de esta práctica.
- Se permite el uso de apuntes de clase, diapositivas, ejercicios prácticos y código fuente. Sin embargo, todo este material debe descargarse antes de comenzar a resolver el enunciado.
- Se permite el uso de Internet exclusivamente para consultar páginas oficiales de Microsoft y Oracle. Sin embargo, cualquier forma de comunicación con otros estudiantes o terceros está estrictamente prohibida.

Puntaje total: 20 puntos

- Cada propuesta de solución a cada pregunta deberá subirse a la plataforma `Paideia` en un archivo comprimido en formato `ZIP`. No se aceptarán los trabajos compactados con otros programas como `RAR`, `WinRAR`, `7zip` o similares. Es importante asegurarse de incluir únicamente el código fuente, excluyendo el código objeto y las librerías.
- Cada archivo deberá tener el siguiente formato de nombre "Lab02_2025_1_CO_PA_PN_PR" donde: **CO** indica: Código del alumno, **PA** indica: Primer Apellido del alumno, **PN** indica: primer nombre y **PR** indica: número de la pregunta pudiendo ser PR igual a P1, P2 o P3 únicamente. De no colocar este requerimiento se le descontará 3 puntos de la nota final. Un ejemplo de formato podría ser el siguiente: "Lab02_2025_1_19941146_Melgar_Héctor_P2".
- En la plataforma `Paideia` se encuentran 3 archivos que puede utilizar en su propuesta de solución: `Principal_P1.cs`, `Principal_P2.cs` y `Principal_P3.cs`, que contienen el código del método `Main` de cada pregunta. El archivo `Principal_P1.cs` contiene el `Main` de la pregunta 1, el archivo `Principal_P2.cs` contiene el `Main` de la pregunta 2 y el archivo `Principal_P3.cs` contiene el `Main` de la pregunta 3.

Cuestionario

- Los objetivos de este laboratorio son:
  - Reforzar los conceptos del curso *Programación 2* relacionados con el Paradigma Orientado a Objetos, con énfasis en los mecanismos de **encapsulamiento**, **herencia** y **polimorfismo**.
  - Implementar propuestas de solución aplicando los mecanismos del Paradigma Orientado a Objetos, utilizando el lenguaje de programación `C#`.
  - Aplicar las mejores prácticas de programación adquiridas a lo largo de la carrera, incluyendo los principios de DRY (*Don't Repeat Yourself*), prevención del hardcoding, nomenclatura semántica y coherente para clases y métodos, así como otros principios de diseño y mantenibilidad del software.

Pregunta 1 (7 puntos)

Un programa académico de estudios desea automatizar el proceso de admisión para determinar qué postulantes han sido admitidos. Para ello, se ha diseñado un conjunto de clases que modelan el proceso de evaluación.

El método `Main` de la clase `Principal` ya ha sido implementado y no puede ser modificado (ver programa 1). En este método, se crean instancias de postulantes y sus respectivas fichas de evaluación. Su tarea consiste en implementar las clases necesarias para que el sistema funcione correctamente.

La clase `Postulante` representa a un postulante que desea ingresar al programa académico. Debe contener los siguientes atributos privados: `paterno`, `materno`, `nombre` y `dni`. Todos ellos son cadenas de caracteres.

La clase `FichaEvaluacion` almacena la información de la evaluación de un postulante. Sus atributos privados son: `candidato`, `fecha_hora`, `evaluacion_expediente`, `evaluacion_entrevista`, `evaluacion_examen` y `estado_candidato`. Donde:

- `candidato`: hace referencia al postulante evaluado.
- `fecha_hora`: indica la fecha y la hora de cuándo se realizó la evaluación.
- `evaluacion_expediente`: almacena el puntaje otorgado por la revisión de documentos. Este es un número entero en el rango [0..25].
- `evaluacion_entrevista`: almacena el puntaje obtenido en la entrevista. Este es un número entero en el rango [0..50].
- `evaluacion_examen`: almacena el puntaje obtenido en el examen de admisión. Este es un número entero en el rango [0..25].
- `estado_candidato`: almacena el estado del postulante según la siguiente enumeración: `ADMITIDO`, `NO_ADMITIDO`, `SIN_EVALUACIÓN`.

La clase `Admision` gestiona la lista de postulantes y los resultados de admisión. Sus atributos privados son: `listaFichas`, `cantidad_admitidos` y `cantidad_postulantes`. Donde:

- `listaFichas`: almacena todas las fichas de evaluación.
- `cantidad_admitidos`: registra cuántos postulantes fueron admitidos.
- `cantidad_postulantes`: indica el número total de postulantes evaluados.

Se le pide que implemente usando el lenguaje `C#` lo siguiente, considerando sobre todo el mecanismo del encapsulamiento:

- (1 punto) Implementar la clase `Postulante` considerando los atributos antes mencionados y utilizando una propiedad para cada atributo, además implemente el constructor con parámetro, el constructor sin parámetro y el constructor copia.
- (1 punto) Sobreescriba el método `ToString` en la clase `Postulante` de forma tal que retorne una cadena de caracteres que incluya los datos del postulante: primero el apellido paterno, luego el apellido materno, seguido por una coma, luego los nombres y finalmente el DNI entre paréntesis. Por ejemplo: `Pérez Deza, Juan Alonso (75355946B)`.
- (1 punto) Implementar la clase `FichaEvaluacion` considerando los atributos antes mencionados y utilizando una propiedad para cada atributo, además implemente el constructor con parámetro, el constructor sin parámetro y el constructor copia.
- (1 punto) Modifique el comportamiento del método `get` de la propiedad vinculada al atributo `estado_candidato` para que antes de retornar el estado, sume todos los puntajes del postulante (evaluación de expediente, evaluación de la entrevista y la evaluación del examen) y si este puntaje es mayor de 75, que es considerado el puntaje mínimo, actualice el estado para `ADMITIDO`, caso contrario, actualice el estado para `NO_ADMITIDO`. Recuerde que no está permitido usar *constantes mágicas*.

- (1 punto) Implementar la clase `Admision` considerando los atributos antes mencionados, además implemente el constructor sin parámetro.
- (1 punto) Implemente en la clase `Admision` el método `agregarFichaDeEvaluacion` que permite agregar una ficha de evaluación en la lista `listaFichas`. Debe considerar que al momento de agregar una ficha debe actualizar la cantidad de postulantes, la cantidad de admitidos, así como el estado de admisión de cada postulante.
- (1 punto) Sobreescriba el método `ToString` en la clase `Admision` de forma tal que retorne una cadena de caracteres que permita imprimir el reporte tal como se presenta en la `Consola`.

```
using System;

namespace Pregunta1
{
    public class Principal
    {
        static void Main(string[] args)
        {
            Postulante postulante = new Postulante();
            FichaEvaluacion ficha = new FichaEvaluacion();
            Admision admision = new Admision();

            postulante.Paterno = "Pérez";
            postulante.Materno = "Deza";
            postulante.Nombre = "Juan Alonso";
            postulante.Dni = "75355946B";
            ficha.Candidato = postulante;
            ficha.Fecha_hora = new DateTime(2025, 02, 20, 14, 00, 00);
            ficha.Evaluacion_expediente = 22;
            ficha.Evaluacion_entrevista = 47;
            ficha.Evaluacion_examen = 18;
            admision.agregarFichaDeEvaluacion(ficha);

            postulante.Paterno = "León";
            postulante.Materno = "Mendoza";
            postulante.Nombre = "Carmen";
            postulante.Dni = "87332141Z";
            ficha.Candidato = postulante;
            ficha.Fecha_hora = new DateTime(2025, 02, 20, 14, 30, 00);
            ficha.Evaluacion_expediente = 12;
            ficha.Evaluacion_entrevista = 22;
            ficha.Evaluacion_examen = 17;
            admision.agregarFichaDeEvaluacion(ficha);

            postulante.Paterno = "Sandoval";
            postulante.Materno = "García";
            postulante.Nombre = "Eric";
            postulante.Dni = "73734226K";
            ficha.Candidato = postulante;
            ficha.Fecha_hora = new DateTime(2025, 02, 20, 15, 00, 00);
            ficha.Evaluacion_expediente = 15;
            ficha.Evaluacion_entrevista = 45;
            ficha.Evaluacion_examen = 23;
            admision.agregarFichaDeEvaluacion(ficha);

            Console.WriteLine(admision);
        }
    }
}
```

Programa 1: clase Principal.cs

Consola

```
PROCESO DE ADMISION: 3 postulantes, 2 admitidos
        LISTA DE ADMITIDOS:
               Pérez Deza, Juan Alonso (75355946B)
               Sandoval García, Eric (73734226K)
```

Consideración 1

En caso requiera que los atributos permitan almacenar valores nulos, deberá añadir el caracter `?` al tipo de dato, por ejemplo:

```
private int? evaluacion_expediente;
private int? evaluacion_entrevista;
```

Consideración 2

El compilador `csc` no acepta la implementación de las propiedades con la notación lambda (`=>`), por lo tanto no la use. Se recomienda que implemente las propiedades de la siguiente manera:

```
public int? Evaluacion_expediente
{
    get {
        return evaluacion_expediente;
    }
    set {
        evaluacion_expediente = value;
    }
}
```

Pregunta 2 (7 puntos)

En la Programación Orientada a Objetos, uno de los problemas más comunes al trabajar con clases que poseen muchos atributos opcionales es la dificultad de crear instancias de manera flexible.

Por ejemplo, consideremos la clase `Cancion`, que almacena los siguientes atributos privados:

- `titulo`: almacena el título de una canción.
- `otroTitulo`: almacena el título alternativo de una canción.
- `interpretes`: almacena la lista de intérpretes de la canción.
- `compositores`: almacena la lista de compositores de la canción.
- `generoMusical`: almacena el estado de la canción según la siguiente enumeración: `FOLKLORE` y `CLASICA`.
- `album`: almacena el nombre del album en donde se grabó la canción.
- `opus`: almacena el indicador de opus de la canción.
- `subOpus`: almacena el indicador de sub opus de la canción.
- `dedicatoria`: almacena la dedicatoria de la canción.

El principal problema con este tipo de clases es que no todas las canciones contienen todos los atributos de la clase `Cancion`, inclusive las del mismo tipo.

- Algunas pueden tener títulos alternativos, pero la mayoría no.
- Los atributos opus y sub opus son comunes en la música clásica, pero no en otros géneros.
- Los intérpretes y compositores pueden ser uno o varios.
- La dedicatoria suele ser un atributo opcional en la música clásica.

Para solucionar este problema, se desea implementar una clase de soporte llamada `CancionBuilder`, esta clase debe permitir la construcción flexible de objetos `Cancion`, de manera que solo se asignen los atributos necesarios para cada caso, sin la necesidad de múltiples constructores o parámetros opcionales desordenados. El código debe funcionar con el programa de prueba (ver programa 2) sin modificarlo.

Se le pide que, usando el lenguaje `C#`:

- (1 punto) Implementar la clase `Cancion` considerando los atributos antes mencionados y utilizando una propiedad para cada atributo, además implemente el constructor sin parámetro. Implemente además el enumerado `Genero`.
- (1 punto) Sobreescriba el método `ToString` en la clase `Cancion` de forma tal que retorne una cadena de caracteres que incluya los datos de la canción que no sean nulos. Vea el resultado de la `Consola` para el formato de la sobreescritura del método.
- (1 punto) Implementar la clase `CancionBuilder`. Esta clase debe contener exactamente los mismos atributos privados que la clase `Cancion`. Implemente los atributos mencionados y utilice una propiedad para cada atributo, además implemente el constructor sin parámetro.
- (2 puntos) Implementar los siguientes métodos en la clase `CancionBuilder` teniendo en consideración que cada uno de ellos retorna la instancia actual de la clase `CancionBuilder` y además setea el valor del atributo.

  a) `ConTitulo`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `titulo` de la clase `CancionBuilder`.
  b) `TambienConocidaComo`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `otroTitulo` de la clase `CancionBuilder`.
  c) `InterpretadoPor`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `interprete` de la clase `CancionBuilder`.
  d) `CompuestoPor`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `compositor` de la clase `CancionBuilder`.
  e) `DelGenero`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `generoMusical` de la clase `CancionBuilder`.
  f) `EnElAlbum`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `album` de la clase `CancionBuilder`.
  g) `IdentificadoConOpus`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `opus` de la clase `CancionBuilder`.
  h) `IdentificadoConSubOpus`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `subOpus` de la clase `CancionBuilder`.
  i) `DedicadoA`: retorna la instancia actual de `CancionBuilder` y actualiza el atributo `dedicatoria` de la clase `CancionBuilder`.

- (2 puntos) Implemente el método `BuildCancion` de la clase `CancionBuilder`. Este método crea una instancia de la clase `Cancion`, asigna cada uno de los atributos de la instancia de `CancionBuilder` a la instancia de la clase `Cancion` y finalmente, retorna la instancia de la clase `Cancion`.

```
1  using System;
2
3  namespace Pregunta2
4  {
5      public class Principal
6      {
7          static void Main(string[] args)
8          {
9              CancionBuilder cancionBuilder = new CancionBuilder();
10             Cancion cancion = cancionBuilder
11                 .ConTitulo("Cuando estoy triste")
12                 .TambienConocidaComo("Cajita de música")
```

```
                .InterpretadoPor("Mercedes Sosa")
                .CompuestoPor("Jose Pedroni")
                .CompuestoPor("Damián Sánchez")
                .DelGenero(Genero.FOLKLORE)
                .EnElAlbum("A que florezca mi pueblo")
                .BuildCancion();
            Console.WriteLine(cancion);
            Console.WriteLine();

            cancion = new CancionBuilder()
                .ConTitulo("Sonata para violonchelo n.1 en fa mayor")
                .CompuestoPor("Ludwig van Beethoven")
                .DelGenero(Genero.CLASICA)
                .IdentificadoConOpus(5)
                .IdentificadoConSubOpus(1)
                .DedicadoA("Federico Guillermo II de Prusia")
                .BuildCancion();
            Console.WriteLine(cancion);
            Console.WriteLine();

            cancion = new CancionBuilder()
                .ConTitulo("Concierto para piano n.1 en do mayor")
                .CompuestoPor("Ludwig van Beethoven")
                .DelGenero(Genero.CLASICA)
                .IdentificadoConOpus(15)
                .DedicadoA("Princesa Barbara Odescalchi")
                .BuildCancion();
            Console.WriteLine(cancion);
            Console.WriteLine();
        }
    }
}
```

Programa 2: clase Principal.cs

Consola

```
TITULO: Cuando estoy triste
TAMBIEN CONOCIDA COMO: Cajita de música
INTERPRETADO POR: Mercedes Sosa
COMPUESTO POR: Jose Pedroni, Damián Sánchez
TIPO: Folklore
ALBUM: A que florezca mi pueblo

TITULO: Sonata para violonchelo n.1 en fa mayor
COMPUESTO POR: Ludwig van Beethoven
TIPO: Clásica
OPUS: 5 SUBOPUS: 1
DEDICADO A: Federico Guillermo II de Prusia

TITULO: Concierto para piano n.1 en do mayor
COMPUESTO POR: Ludwig van Beethoven
TIPO: Clásica
OPUS: 15
DEDICADO A: Princesa Barbara Odescalchi
```

Pregunta 3 (6 puntos)

En nuestro país, una de las diferencias entre una boleta de venta y una factura es que, en el caso de la factura, se debe incluir la razón social del cliente, el cual debe ser una persona jurídica, junto con su RUC (Registro Único del Contribuyente).

Para gestionar estos documentos, se cuenta con la siguiente estructura de clases:

- `Comprobante`: clase base que gestiona una lista de detalles de compra mediante la clase `ComprobanteDetalle`.
- `ComprobanteDetalle`: representa cada ítem de la compra y contiene los siguientes atributos: número del ítem, la descripción, la cantidad, el precio y el subtotal.

- `Factura`: Clase derivada de `Comprobante` que extiende su comportamiento, incorporando la impresión de los datos del cliente (razón social y RUC).

Además, la clase `Comprobante` tiene sobreescrito el método `ToString` para imprimir correctamente la información del comprobante, tal como se muestra en el código del programa de prueba (ver programa 3).

Se le pide implementar las clases mencionadas usando el lenguaje `C#` aplicando buenas prácticas de programación orientada a objetos, con especial énfasis en herencia y polimorfismo. Además considere que: no puede modificar el código del programa de prueba (ver programa 3), debe optimizar el código evitando repeticiones innecesarias e implementar correctamente herencia y polimorfismo para extender el comportamiento de `Comprobante` .

- (2 puntos) Implementación de la clase `Comprobante`.
- (2 puntos) Implementación de la clase `ComprobanteDetalle`.
- (2 puntos) Implementación de la clase `Factura`.

```
using System;

namespace Pregunta3
{
    public class Principal
    {
        static void Main(string[] args)
        {
            Comprobante comprobante = new Comprobante();
            comprobante.agregarDetalle("Polo azul", 2, 56.99);
            comprobante.agregarDetalle("Blue Jean", 1, 99.45);
            Console.WriteLine(comprobante);
            Console.WriteLine();

            comprobante = new Factura("10236786549", "Asociación Programación 3");
            comprobante.agregarDetalle("Polo azul", 2, 56.99);
            comprobante.agregarDetalle("Blue Jean", 1, 99.45);
            Console.WriteLine(comprobante);
        }
    }
}
```

Programa 3: clase Principal.cs

Consola

```
BOLETA DE PAGO
DETALLE:
        No Desc.     Cant. Precio IGV   subTotal
        01 Polo azul  02   56,99  18,0 % 134,50
        02 Blue Jean  01   99,45  18,0 % 117,35
TOTAL: 25.184,7 %

FACTURA
CLIENTE:
        RUC: 10236786549
        Razón Social: Asociación Programación 3
DETALLE:
        No Desc.      Cant. Precio IGV   subTotal
        01 Polo azul   02   56,99  18,0 % 134,50
        02 Blue Jean   01   99,45  18,0 % 117,35
TOTAL: 25.184,7 %
```

Profesores del curso: Freddy Paz, Heider Sánchez, Andrés Melgar, Eric Huiza

Pando, 2 de abril de 2025